Partially Adaptive Optimization Algorithm for Deep Convolution Neural Networks to achieve faster convergence and a better generalization

Siddharth Nijhawan
UNI: sn2951
sn2951@columbia.edu

Sushant Tiwari
UNI: st3425
st3425@columbia.edu

Hari Nair
UNI: hn2388
hn2388@columbia.edu

Aman Anand
UNI: aa4821
aa4821@columbia.edu

Abstract

Optimization algorithms used for convergence of the objective function in deep neural networks are majorly designed for the purpose of better generalization to the unseen data as well as faster descent to the optimum. Most of the optimization techniques focus on keeping the learning rate adaptive and using the momentum in the form of exponential weighted average descent to improve the quality of the descent. Adaptive moment estimate (Adam) optimizer, which is the most widely used optimization algorithm in the world of deep learning, despite having faster convergence, suffers from poor generalization to unseen data. We,in this paper, with our mathematical analysis and detailed experimentation, show that no compromise is required with the quality of convergence for achieving better generalization. We implement an optimization approach called, Partially adaptive momentum estimation (Padam) method, which adjusts the adaptive term for the learning rate in a way to achieve generalization comparable to Stochastic Gradient Descent (SGD) with momentum and also converges in a better fashion as compared to the adaptive gradient optimization techniques. Throughout our experimentation and detailed analysis, we compare the performance of Padam by varying the additional adaptive hyperparameter p as well as compare the performance of padam optimization algorithm with respect to other optimization techniques such as SGD with momentum, Adadelta, Adam, AdamW and Nadam over CIFAR-10 and CIFAR-100 datasets by using state-of-the art neural network architectures like ResNet, VGGNet and Wide-ResNet. Through this paper, we also prove that we achieve better optimization as we approach the optimum loss for non-convex objectives.

1. Introduction

Contrary to the widely accepted notion of the superiority of Adam optimizer, SGD with momentum has a more effective approach towards reaching the steady-state value. Optimization algorithms which change their learning rate in accordance with the current position, tend to perform absurdly on unseen dataset. Although, initial rate of convergence for these adaptive learning rate algorithms is faster, this comes as a trade-off with their overall performance on massive datasets. In some cases, it is observed that the model doesn't converge altogether when optimum sets of hyperparameters are not used for training the neural network.

Adagrad optimization accumulates the gradient and since we sum all the previous gradients in calculating the adaptive term for the learning rate, there is no significant movement achieved after a few epochs. Also, Adagrad doesn't include an exponential moving average term for the momentum and hence the batch-wise gradient approach for Adagrad has zig-zags and randomness. However, Adagrad was observed to show inferior performance in instances with dense gradients or non-convex objective functions. RMSProp assigns lesser weights to the accumulated gradient and hence improves the learning rate adaptive term but still has a zig-zag convergence due to the lack of exponential moving average term for the momentum.

Adam takes care of both the step-size for the gradient descent as well as the exponential moving average for the momentum and hence is a huge improvement over other adaptive optimization techniques. Adam suffers from a common problem experienced with the Exponentially Weighted Average where there is some information loss for larger gradients. AMSgrad, resolves this issue by calculating the maximum of previous second order gradient moment with the current updated value of the moment. This prevents the information loss for higher gradients by keeping the adaptive term for the learning rate higher. This also solves the convergence problems encountered in the Adam optimizer. Other improvements in optimization include the use of Strongly Convex Adagrad and Strongly Convex RMS which impose upper bounds (logarithmic) for the functions which are convex.

SGD with momentum, although is slow to begin with, it ultimately achieves a higher test accuracy as compared to the above optimization techniques. SGD with momentum does not have a second order adaptive moment term in the denominator of the parameter update function and hence has a consistent learning rate for every parameter update. However, for the adaptive optimization techniques, we observe a higher learning rate for some points. To compensate for these enlarged step-size, we need to choose a smaller base learning rate. This poses a huge problem in the gradient descent approach as after after successive descents, the second order moment term keeps on accumulating, thereby making the effective learning rate smaller. Due to this problem, parameter updates might cease to take place after certain epochs. We refer to this problem as the *learning rate conundrum*.

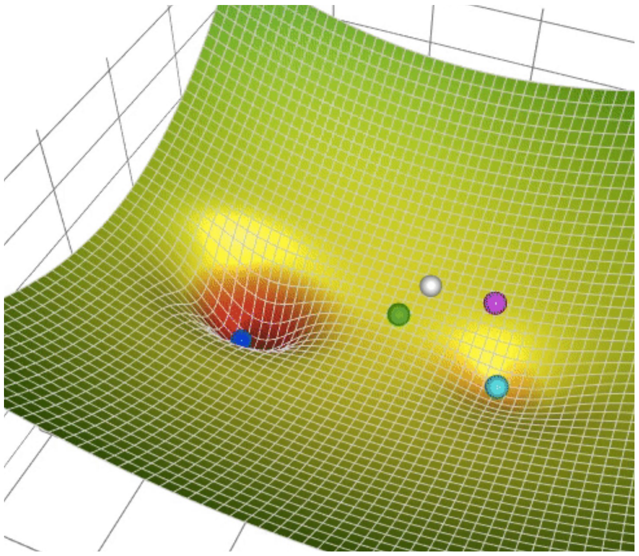

Figure 1. Approach to global and local optimums during descent [17]

The aforementioned reasons point to the fact that there is a scope to design a dynamic optimization algorithm that lies between these two extreme approaches wherein, by the introduction of an adaptive hyperparameter, we can control the convergence and also achieve model generalization as good as that achieved by SGD with momentum. So, the followed approach adjusts the exponent/power of the adaptive learning rate term (also known as normalization factor) by choosing an intermediate hyperparameter p value lying in between the two extremes (0 in case of SGD with momentum and 0.5 in case of Adam optimizer). This will lead to lower successive reduction of effective learning rates and hence we can choose a higher initial or base learning rate solving the *learning rate conundrum*. Previous studies which have focused on solving these issues involve highly complex architectures and optimization algorithms. Despite this, for irregular datasets, the model is not able to generalize as well as the SGD optimization algorithm. Besides, these optimization techniques do not guarantee a convergence for non-convex objective functions which is a major issue as in practical scenario, we come across majority of the functions which do not exhibit convexity.

Our proposed algorithm implementation guarantees the convergence to an optimum even for the non-convex objective functions. By choosing the optimum set of hyper parameters as well as the p value, we achieve much faster convergence rate as compared to the more latest adaptive optimization algorithms besides achieving the baseline test objective loss (comparable to the SGD with momentum method). Our work treats the additional adaptive learning rate term in a continuous manner and searches for the optimum p value between the two extremes.

To summarise our methodology, we implement an algorithm which improves over the initial convergence rate as compared to the SGD with momentum algorithm. This also produces better generalization over the test dataset and hence achieves higher accuracy as compared to the adaptive gradient methods. By changing the value of the hyperparameter p, used in the adaptive learning term having the second order moment, we achieve the optimum in the above two performance metrics. Our implementation also solves the problem of diminishing learning rate conundrum with the help of an optimum p-norm (pth exponent) which gives a leeway to the model in terms of initial learning rate value selection. We optimize different combinations of hyperparameters for widely used optimization techniques and compare their performance with our implemented algorithm.

Also, through our proposed implementation, we propose guaranteed model convergence for non-convex objective functions and derive the model complexity for convergence:

$$O(\frac{k^{\frac{1}{2}}}{I^{\frac{3}{4}-\frac{r}{2}}} + \frac{k}{I})$$

where, k = model dimension, I = number of iterations used in Padam, r = gradient (stochastic) growth rate. $0 \leq r \leq 0.5$. When r is strictly lesser than 0.5, we achieve a better computational complexity and faster convergence with Padam as compared to SGD with momentum.We test our algorithm with the image classification data CIFAR-10 and CIFAR-100 for different optimization algorithms over 3 neural network architectures ResNet, VGGNet and Wide-ResNet to extensively analyse the robustness of Padam.

This paper is structured as follows: Section 2 will cover the related work and existing methodology in the above domain. Section 3 will give a brief overview on the commonly observed optimization techniques. Section 4 will give basic information regarding the data and neural network architectures used for experimentation and detailed information about the implementation including convergence analysis for non-convex optimization objective functions. Section 5 provides details about our analysis and the various experiments conducted on the image classification datasets for various neural network architectures. Section 6 includes the results and inferences and also comperes performance with the previous implementation. Section 7 summarizes the work and provides insight into the future scope in this area.

2. Related Work

Optimization of objective function has been a hot topic for many years now. Most of the existing methodologies and previous studies aim at improving the approach towards the gradient descent. As we have seen, there is a trade-off between the transient response with respect to the batch-training over epochs and the final accuracy achieved on the unseen test data. These optimization algorithms are improvements over Adam optimizer which already had the terms for variable adaptive learning rates and the momentum.

Zhang et al. [2] presents an algorithm that proposes improvements in Adam by tweaking the adjustable terms for momentum and learning rate for each present state is changed only in accordance with the gradient and thus achieving a normalized direction maintaining version (ND-Adam). Keskar et al. [3] proposed that for massive datasets requiring complex objective functions, SGD with momentum had a better performance than Adam optimizer. Before this study, the primary goal for deep learning researchers, was focused on improving the model's convergence rates during its training with pre-trained classic neural network architectures. But, more recent studies have started focusing on improving the robustness and model accuracy for the unseen data. Also, earlier studies such as Reddi et al. [4], Kingma et al. [5], Mukkamala et al. [6] and Duchi et al. [7] covered optimization methodologies utilizing gradient descents in various forms mostly for convex objective functions.

Recent studies have started considering more practical scenarios wherein non-convex objectives have to be optimized. Because of stochasticity involved in the Stochastic Gradient methods, there is randomness involved in the model parameters as to how they change with successive batch (or individually). Hence, later research focused on studying the effects of deterministic and stochastic training data selection. Basu et al. [8] studied the deterministic convergence for the Adam and RMSProp optimization and analysed these gradient approaches with respect to the stochastic optimization techniques. Non-convex optimization was studied by Li et al. [9] with AdaGrad algorithm. This study compared the performance of the algorithm under convex and non-convex scenarios. However, we know that AdaGrad has limitation because even for convex optimization objectives as it doesn't involve the calculation of exponential moving averages for the gradient descent approach. A study by Juntang et al. [10] proposed to adjust the learning rate based on the direction of the current gradient. According to this study, if the gradient observed for an observation, has a large deviation with respect to the prediction, that particular observation is discarded and a small step is taken towards convergence. Similarly, if the gradient observed for an observation, has a negligible deviance with respect to the prediction, then that observation is *trusted* and a large step is taken towards convergence. Wu et al. [11] experimented with changing the learning rates after certain epochs and with the learning rate schedules. Xu et al. [12] analysed the effect of stochasticity on geometric convergence for the diminishing learning rate. Ge et al. [13] analyzed the effect of scheduling learning rate for quadratic or second-degree functions.

There are studies by Loshchilov et al. [14], Luo et al. [15] and Zaheer et al. [16] which have focused on changing the adaptive learning rate terms for better convergence in the adaptive gradient methods. These works had an objective similar to the methodology adopted by Padam optimization but focused on changing the constant *epsilon* in the term. Varying this additive parameter has a negligible effect in controlling the convergence rate and hence, our implementation is a better suggestion. Further research by Loshchilov et al. [14] proposed using AdamW which decouples the updation of gradient from the weight decay in every iteration and hence achieving a lower test accuracy and better generalization.

AdamW [Loshchilov and Hutter, 2019] [14] proposed to fix the weight decay regularization in Adam by decoupling the weight decay from the gradient update and this improves the generalization performance of Adam.

This work aims to build on the results achieved by Jinghui et al. [1] which introduced the Padam algorithm as an improvement over existing adaptive optimization techniques and validated its performance on large image

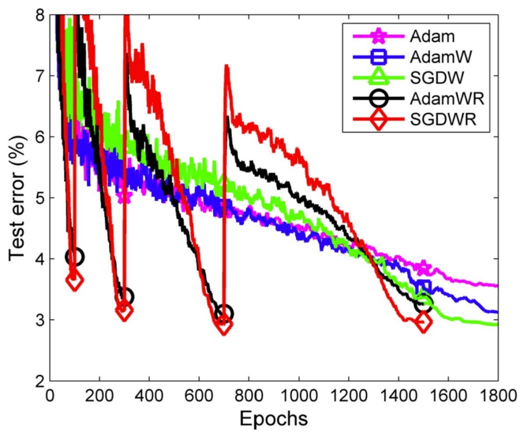

Figure 2. Top-1 test error on CIFAR-10 by Loshchilov et al. [14]

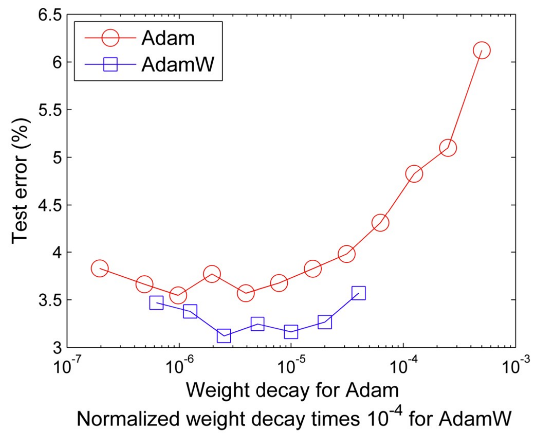

Figure 3. Weight Decay for Adam by Loshchilov et al. [14]

datasets such as CIFAR-10 and CIFAR-100 and LSTM language models.

For Padam optimizer, Jinghui et al. [1] analysed the test error over different p values [$\frac{1}{16}$, $\frac{1}{8}$ and $\frac{1}{4}$]. Learning rates were decreased by 0.1 after 100 and 150 epochs. Lower p values had a performance similar to SGD with momentum optimizer which had a slow initial convergence rate with better generalisation as compared to the case when p was higher. This observation was observed on both the image classification datasets as well as the LSTM language models. However, variation of p didn't show a significant effect on language model output as compared to the image classification output.

The above study by Jinghui et al. [1] used SGD with momentum, Adam, Amsgrad, AdamW, Yogi, AdaBound and Padam as optimizers on VGGNet, ResNet and WideResNet neural network architectures.On VGGNet

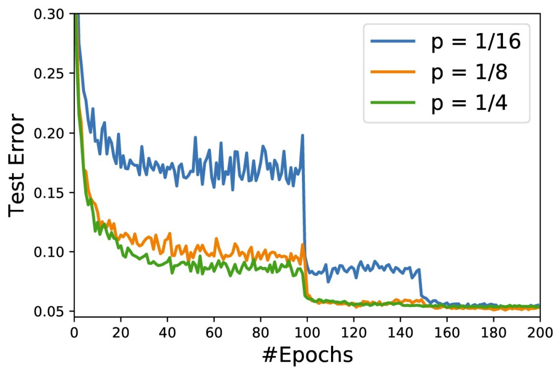

Figure 4. Padam with ResNet on CIFAR-10 by Jinghui et al. [1]

Models	SGD-Momentum	Adam	Amsgrad	AdamW	Yogi	AdaBound	Padam
VGGNet	93.71	92.21	92.54	93.54	92.94	93.28	**93.78**
ResNet	**95.00**	92.89	93.53	94.56	93.92	94.16	94.94
WideResNet	95.26	92.27	92.91	95.08	94.23	93.85	**95.34**

Figure 5. Test accuracy on CIFAR-10 by Jinghui et al. [1]

and for CIFAR-10, Padam achieved the best accuracy of 93.78%. Similarly, Padam achieved the highest accuracy of 95.34% on WideResNet whereas SGD with momentum scored by attaining the highest accuracy of 95% on ResNet architectures. Padam also achieved decent accuracy of 61.53% and 58.48% on 2-Layer and 3-Layer LSTMs respectively.

Although, almost similar accuracy values were achieved for all other optimizers in case of language models indicating the fact that the Padam is much more effective for image classification and other complex non-sequential datasets. The above study also trained the ResNet and VGGNet model architectures on ImageNet dataset in which Padam optimizer again achieved the highest top-1 and top-5 accuracy of 74.04% and 91.93% respectively. Padam outperformed other optimization algorithms with 89.47% top-5 test accuracy on ResNet architecture whereas SGD with momentum overshadowed other optimizers including Padam on achieving top-1 test accuracy with a value of 70.23%

3. Overview of Optimization Algorithms

3.1. SGD with momentum

Stochastic Gradient Descent uses a single training sample to train which leads to a faster convergence as compared to the normal Gradient Descent which uses all the training points and parameters for a single update. SGD with momentum uses exponential moving average to avoid the zig-zag motions of parameters nullifying the effect of randomness in descent due to stochasticity. This leads to a better and balanced approach in achieving the optimum.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Figure 6. Update equations using SGD with momentum [21]

3.2. AdaGrad

AdaGrad optimization uses adaptive learning rates with smaller step sizes for the feature values which are frequent and larger step sizes for the features which are infrequent. AdaGrad eliminates the requirement for manual adjustment of the learning rate and hence improves the descent approach. The disadvantage of using this optimizer is that it keeps on accumulating the gradients and hence for successive steps, the learning rate diminishes. This leads to a minimal parameter update as the point comes in the vicinity of the optimum.

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

Figure 7. Update equations using AdaGrad [21]

3.3. AdaDelta

Adadelta optimization also uses adaptive learning rates with smaller step sizes for the feature values which are frequent and larger step sizes for the features which are infrequent but differs in robustness to the Adagrad optimizer. It doesn't keep on accumulating the previous gradient unlike Adagrad. There is no need to set the intial or base learning rate in Adadelta. The stochastic gradient sum in Adadelta is the repeated average of diminishing values of the squared gradients for the past iterations.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\Delta\theta_t = -\eta \cdot g_{t,i}$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Figure 8. Update equations using AdaDelta [21]

3.4. RMSProp

RMSprop improves the adaptive learning rate term used in AdaGrad by dividing it with the exponential moving average of the squared gradients. We normally take 0.9 as the coefficient of the exponential moving average term for the squares of the past accumulated gradients and 0.1 as the coefficients of the current gradient at time t. The vectors are updated such that the learning rate does not diminish as the point approaches the optimum.

$$E[g^2]_t = 0.9 E[g^2]_{t-1} + 0.1 g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Figure 9. Update equations using RMSProp [21]

3.5. Adam

Adam is an improvement over RMSprop as in addition to using "intelligent" updation of learning rates, it also uses the exponential moving average for the momentum term. Therefore, Adam optimizer combines the adaptive learning rate calculation used in RMSprop with the momentum calculation used in SGD with momentum and has been one of the most widely used optimizers in training the deep neural networks.In Adam, initial baseline value of the learning rate has to be kept low in order to avoid the diminishing learning rate conundrum. This poses the problem of diminishing learning rate as the point approaches the extremum.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$$

Figure 10. Update equations using Adam [21]

3.6. Nadam

Nadam uses the Nesterov Adaptive Gradient (NAG) approach in combination with the above discussed Adam optimization technique. Nadam update involves updating the gradient after the momentum parameter i.e. momentum vector is varied in current iteration and then the gradients are updated to achieve better generalization in the direction of the optimum point.

$$g_t = \nabla_{\theta_t} J(\theta_t)$$
$$m_t = \gamma m_{t-1} + \eta g_t$$
$$\theta_{t+1} = \theta_t - m_t$$

$$\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t)$$

$$g_t = \nabla_{\theta_t} J(\theta_t)$$
$$m_t = \gamma m_{t-1} + \eta g_t$$
$$\theta_{t+1} = \theta_t - (\gamma m_t + \eta g_t)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\left(\frac{\beta_1 m_{t-1}}{1-\beta_1^t} + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\left(\beta_1 \hat{m}_{t-1} + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right)$$

Figure 11. Update equations using Nadam [21]

3.7. AMSgrad

AMSgrad, resolves the issue of diminishing learning rate experienced in Adam by calculating the maximum of previous second order gradient moment with the current updated value of the moment. This prevents the information loss for higher gradients by keeping the adaptive term for the learning rate higher.

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$$

$$\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$$
$$\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}m_t$$

Figure 12. Update equations using AMSGrad [21]

4. Dataset, Neural Network Architectures and Proposed Methodology

This section gives details of the data and the neural network architectures used for implementation of Padam optimization. It also discusses our proposed optimization methodology and optimization algorithm.

4.1. Data

CIFAR-10 and CIFAR-100 datasets are used in implementation of proposed algorithm. These datasets are widely popular as the baseline datasets for testing the performance of the neural networks across different architectures.

4.1.1 CIFAR-10

There are 60,000 total color images in CIFAR-10 out of which 10,000 are the test images and remaining 50,000 are the training images. There are total 10 classes and each class is having 6000 images. Each color image is of the dimension 32x32.There is no overlap between any class.

4.1.2 CIFAR-100

There are 60,000 total color images in CIFAR-100 out of which 10,000 are the test images and remaining 50,000 are the training images. There are total 100 classes and each class is having 600 images. Each class consists of 500 training images and 100 testing images. Each color image is of the dimension 32x32.There is no overlap between any class.

4.2. Neural Network Architectures

There are 3 different Neural Network architectures used for implementation of the proposed algorithm. These are VGG-16, ResNet-18 and Wide ResNet.

4.2.1 VGG-16

There are total 16 convolution layers having weights in VGG-16. Instead of increasing number of parameters and trying to finetune these paramters, VGG-16 focuses on using odd sized 3x3 kernels with a unit stride (stride = 1). Maxpool layers use the kernels of size 2x2 each having strides of 2. Entire architecture comprises of convolution layer succeeded by a maxpooling layer. There are 2 fully connected layers at the end followed by Softmax activation function for multi-class classification.

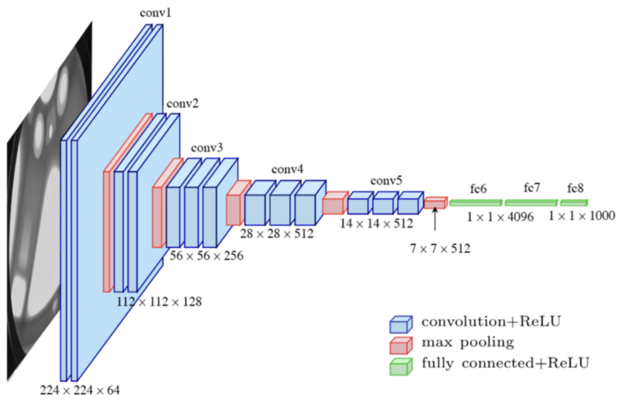

Figure 13. VGG-16 Architecture [18]

4.2.2 ResNet-18

ResNet-18 architecture enables training of deeper and complicated neural networks without undergoing any performance degradation by employing residual network architecture consisting of skip connections and thereby solving the problem of vanishing/diminishing gradients during backward propagation. It allows to map the input as the identity function to the successive layers and hence, prevents any information loss thereby guaranteeing at least input level performance at the output due to the identity mapping.

4.2.3 Wide ResNet

The difference between the ResNet and Wide ResNet is that the later is having higher number of channels with the similar architecture. This allows the neural network to have fewer number of layers so that it can be less deeper resulting in lesser time required for the model to train. If the wide Neural Network is having widening-factor w and depth n then it can be represented as WRN-n-w where WRN stands for the Wide Residual Network.

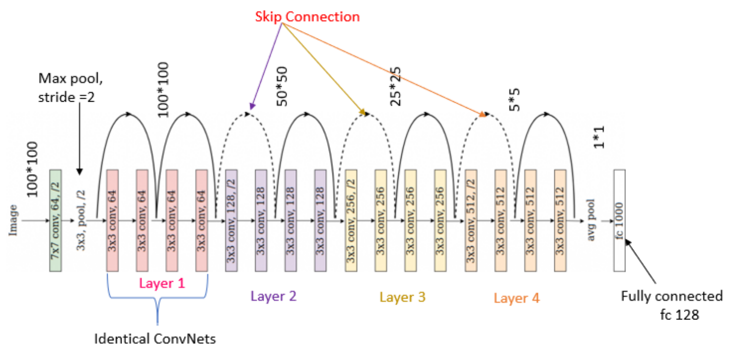

Figure 14. ResNet-18 Architecture [19]

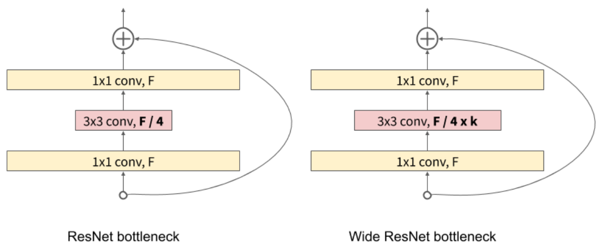

Figure 15. Residual Unit modification in Wide ResNet Architecture [20]

4.3. Proposed Methodology and Algorithm

This section analyzes our proposed implementation methodology and algorithm for the optimization of objectives of the deep neural networks. It introduces a simple learning rate adaptive hyperparamter p for dynamically varying the level of step size in between the 2 already seen extremes of SGD with momentum and Adam / AMSGrad. Since, we add an extra hyperparamter p in the optimization techniques (which are on the same lines as Adam), this technique is called Padam. (p for partial).

Stochastic gradient is calculated with respect to the update parameter and is denoted by g_t. Learning rate is denoted by η. 1st order moments of the above calculated stochastic gradients are denoted by m_t and 2^{nd} order moments of the gradients are denoted by v_t. The parameter which needs to be updated with each iteration is denoted by θ. The parameter β_1 decides the relative weighted importance to the previously calculated 1^{st} order moment m_{t-1} compared to the stochastic gradient at time t. The parameter β_2 decides the relative weighted importance to the previously calculated 2^{nd} order moment v_{t-1} compared to the stochastic gradient at time t. Prediction of the value for the 2^{nd} order moment is denoted by $\hat{v}$.

Algorithm 1 Padam: p - adaptive moment estimation

```
**Input:** start point $x_1 \in X$; learning rate parameters $\beta_1, \beta_2$, p $\in [0, \frac{1}{2}]$
set $v_0 = 0$, $\hat{v}_0 = 0$, $m_0 = 0$
**for** $i = 1, ..., I$ **do**
    g$_i = \nabla f(x_i, c_i)$
    $m_i = \beta_1 m_{i-1} + (1 - \beta_1) g_i$
    $v_i = \beta_2 v_{i-1} + (1 - \beta_2) g_i^2$
    $\hat{v}_i = max(\hat{v}_{i-1}, v_i)$
    x$_{i+1} = x_i - \eta_i \frac{m_i}{(\hat{v}_i + \varepsilon)^p}$
**end for**
**Output:** Choose x$_{out}$ from $x_i$, $2 \leq i \leq I$ with probability $\frac{\eta_{i-1}}{(\sum_{i=1}^{1-1} \eta_i)}$
```

Step-wise implementation for the Padam algorithm is shown below:

Step-1: Initialize the value of the 1^{st} order and 2^{nd} order moments as 0. We name this as 0^{th} iteration. Value for the prediction of the initial 2^{nd} order moment i.e. $\hat{v}$ is also set to 0 in this base iteration. *p* hyperparameter is varied between 0 and 0.5. When p = 0, learning rate doesn't adapt similar to what happens in SGD with momentum. When p = 0.5, the algorithm reduces to AMSGrad, and in this scenario, the generalization error is high. *p* value is varied between 0 and 0.5 at discrete steps to test the accuracy so that low generalization error in addition to high convergence rate is achieved.

Step-2: Iterate the algorithm over T iterations. For the t^{th} iteration, first stochastic gradient is calculated with respect to the parameter theta at time t. Note: epsilon is added to ensure that the learning rate adaptive parameter does not reduce to 0 otherwise it will lead to the divide-by zero error for the effective learning rate. This ϵ is a small constant value and doesn't have any impact in the stochastic gradient calculation.

Step-3: Calculate the value of 1^{st} and 2^{nd} order moments m_t and v_t respectively. β_1 and β_2 are constants and are pre-decided. Moments for the previous iteration at time $t-1$ (m_0 and v_0 for the 1^{st} iteration) along with the above calculated stochastic gradients are used in calculation of the current moments.

Step-4: Calculate the prediction of the learning rate adaptive term phat for the current iteration at time t. Note: We add in ϵ term to this prediction before taking the p^{th} root in this term. To solve the diminishing learning rate problem encountered in Adam (as done in AMSGrad), the prediction is calculated by taking the maximum of the previous prediction at time $t-1$ and the value of the 2^{nd} order moment calculated in Step-3.

Step-5: Update the parameters for the next iteration t+1 by using the values calculated in Step-4 and Step-3. Repeat this process for T iterations.

Introduction of the adaptive hyperparameter *p* effectively solves the problem of learning rate conundrum described above. To explain this problem with respect to the above implementation, notice that the term $(\hat{v}_i + \varepsilon)^p$, controls the values of the effective learning rate for successive iteration. For smaller values of the prediction $\hat{v}$, when *p* is close to its upper bound value of 0.5, the effective learning rate will be high for certain points. This will negatively impact the algorithm's descent as for these points, the randomness can be high leading to poor convergence and decrease in test accuracy. To overcome this, the initial values of the base learning rate is set to a smaller value for all the optimizers which involve adaptive learning rates like Adam, AMSGrad, Nadam. With successive iterations, effective learning rate keeps on decreasing as the point approaches the optimum. Lower step sizes for the learning rate prevent the deeper neural network to update the parameters and hence neural network cannot learn any further for successive iterations. Padam solves this problem by allowing to vary the value of the adaptive hyperparameter *p*. When *p* is reduced to a value less than 0.5, the term $(\hat{v}_i + \varepsilon)^p$ decreases. This counter balances or nullifies the effect of diminishing learning rates. Hence, the model will have a smoother as well as a faster descent to the optimum point. This guarantees a better generalization as compared to Adam and AMSGrad with the same convergence rates as achieved in the SGD with momentum.

Although a step change in the learning rate can be achieved by employing learning rate schedules to decrease the learning rate by a factor after certain iterations but we won't be able to see the exact effect of variation in the adaptive hyperparameter *p* in such case. The main aim here is to devise and implement an algorithm which solves the trade-off for the generalization error and convergence rates and hence, we will keep the learning rate schedule fixed for all the optimization algorithms.

Kingma et al. implemented a version of Adam which first mentioned the p normalization of the adaptive term and then deduced AdaMax optimizer by changing the p value to infinity. The algorithm which we have implemented in this work and through our analysis and experiments, is different from that implemented in the above paper as in our implementation, the adaptive hyperparameter p is changed or scaled which in turn scales the net effective learning rate.

4.4. Operational Complexity of Padam for Non-Convex Objectives

Given that the objective is non-convex, calculation of the stochastic gradient is not straightforward. Also, we cannot directly calculate this gradient because of the presence of stochasticity or randomness in selection of a point per iteration as the gradient will only be valid for that chosen point in that iteration. We can only estimate (unbiased) the gradient and for that we have to assume following constraints:

Constraint-1: If, we take $f(x)$ as the expectation of $f(\theta, c)$ over the entire c, then infinity norm of gradient of this function will be bounded by an upper limit of K_∞ such that:

$$f\left(\theta\right) = E_c f\left(\theta, c\right)$$

$$\left\|\nabla f\left(\theta; C\right)\right\|_\infty \leq G_\infty$$

Reasoning: We know that property for the l2 norm implies that it is limited by an upper bound of K-2 as shown in Reddi et al, this property is a stronger limitation than that of the infinite norm constraint (constraint-1) mentioned above. l2 norm would be greater than the infinity norm as it satisfies the following equation:

$$\left\|\nabla f\left(\theta; C\right)\right\|_\infty \leq \left\|\nabla f\left(\theta; C\right)\right\|_2 \leq \left\|\sqrt{B}\nabla f\left(\theta; C\right)\right\|_\infty$$

This shows that the l2 norm is stronger than infinite norm by a factor of $\sqrt{B}$ We can see that if we replace the infinity norm by the l2 norm in the above inequality and K_∞ by K_2 then it is proved that the infinity norm of gradient of f(x) will have a maximum value or the upper bound of K_2

Constraint-2: According to the function smoothness property, the function is defined by $f\left(\theta\right) = E_c f\left(\theta, c\right)$ satisfies the L-Lipschitz property for the gradients which is given by:

$$\left|f\left(\theta_1\right) - f\left(\theta_2\right) - \left\langle \nabla f\left(\theta_2\right), \theta_1 - \theta_2 \right\rangle\right| \leq \frac{L}{2}\left\|\theta_1 - \theta_2\right\|_2^2$$

This is more specific form of a general L-Lipschitz inequality which is given as:

$$\left\|\nabla f\left(\theta_1\right) - \nabla f\left(\theta_2\right)\right\|_2 \leq L\left\|\theta_1 - \theta_2\right\|_2$$

4.4.1 Analysis for the convergence complexity for non-convex functions

We know that $\beta_2^{2p} > \beta_1, 0 \leq s \leq 0.5$. Also, $q = 4p - 1$ for the values of (4p-1) greater than 0. This implies that the average of the gradient's l2 norm is limited by the upper-bound which is given by:

$$E(\left\|\nabla f\left(\theta\right)\right\|] \leq \frac{A}{I\eta} + \frac{Bd}{I} + \frac{c\eta d G_\infty^{1-q}}{I^{(1-q)(0.5-s)}}$$

where

$$A = 2G_\infty^{2p}\Delta f$$

$$B = \frac{4G_\infty^{2+2p}E\left[\left\|\widehat{v_1}^{-p}\right\|\right]}{d\left(1 - \beta_1\right)} + 4G_\infty^2$$

Here, we can see a clear relationship between the model hyperparameters, the adaptive parameter as well as number of iterations. The terms B and C do not have any dependency on the number of iterations in the descent and hence we can segregate the constants from the number of trials which later can be written in the form of convergence complexity. The constant s will attain its upper bound of 0.5 when there is no sparsity in the stochastic gradients. In sparse condition, s is limited between the range of 0 to 0.5.

Assuming that 4p-1 is greater than 0, for Padam, the expectation of the gradient's l2 norm is limited by the upper-bound which is given by:

$$C = \frac{4LG_\infty^{1+q-2p}}{\left(1-\beta_2\right)^{2p}} + \frac{8\lfloor G_\infty^{1+q-2p}\left(1-\beta_1\right)}{\left(1-\beta_2\right)^{2p}\left(1 - \frac{\left(\beta_1\right)}{\beta^{2p}}\right)}\left(\frac{\beta_1}{1-\beta_1}\right)^2$$

Under an optimum scenario, we choose the learning rate for Padam optimizer as:

$$E[\left\|\nabla f\left(\theta\right)\right\|_2^2 \leq \frac{A}{I\eta} + \frac{Bd}{I} + \frac{C'\eta d G_\infty}{I^{0.5-s}}$$

$$C' = \frac{4lG_\infty^{1-2p}}{\left(1-\beta_2\right)^{2p}} + \frac{8LG_\infty^{1-2}P_{1+p_1},}{\left(1-p_2\right)^{2p}\left(1 - \frac{\beta_1}{\beta_2^{2p}}\right)}\left(\frac{\beta_1}{1-\beta_1}\right)^2$$

$$\eta = \phi \left(d^{0.5} I^{0.25+0 \cdot 5s}\right)^{-1}$$

Then by above equation for the expectation of the gradient's l2 norm for Padam, we obtain the following equation:

$$E\left[\|\nabla f(\theta)\|_2^2\right] = O\left(\frac{d^{0.5}}{I^{0.75-0.5s}} + \frac{d}{I}\right)$$

Even in the scenario when the gradient is not sparse and the value of the parameter $s = 0.5$, we substitute the value of $s = 0.5$ to the above equality. Hence, final worst-case complexity for the Padam optimizer for the non-convex objective is:

$$E\left[\|\nabla f(\theta)\|_2^2\right] = O\left(\sqrt{d/T} + d/T\right)$$

Hence, the final complexity of the Padam optimizer with respect the number of iterations for non-convex objectives is proportional to $O\left(1/\sqrt{T}\right)$

5. Experiments and Observations

We performed experiments on different combinations of adaptive hyperparameter by varying it from 0 to 0.5 in discrete steps for the 3 neural network architectures. Initially performance of the Padam optimizer is checked on ResNet architecture for 3 different values of p [0.25,125, 0.0625 and 0.5]. When p=0.5, the Padam optimizer will show a similar performance to AMSGrad optimizer. We find an optimum value of p for the Padam optimizer as p = 0.125 and then train the image dataset on the other two network architectures: VGGNet and Wide ResNet with this optimum value of p. In the part-2 of our experiments, we compare performance of the Padam optimizer having the optimum p value, with the other standard optimizers such as Adam, AdamW, SGD with momentum and AdaDelta. The above steps are repeated for both the image classification datasets : CIFAR-10 and CIFAR-100.

Here, our main objective is to optimize the proposed algorithm by checking its performance on different p values. We also aim to compare the performance of all modern sets of optimizers with the Padam optimizer. Because of above two objectives, we keep other hyperparameters like learning rate decay, learning rate scheduler value, batch size etc. constant.-Otherwise if these model specific parameters are varied then we would not be able to compare the response of all the optimizers on the base set of conditions.

Model hyperparameters are initialized with the following values: validation train split ratio = 0.9, epochs = 50 or 100, learning rate scheduler milestones = [40,80], learning rate scheduler gamma = 0.1, batch size = 32 (relevant according to the chosen optimizer algorithm) and momentum = 0.9.

.

5.1. Padam Optimizer-ResNet-18 Architecture on CIFAR-10 for multiple values of p

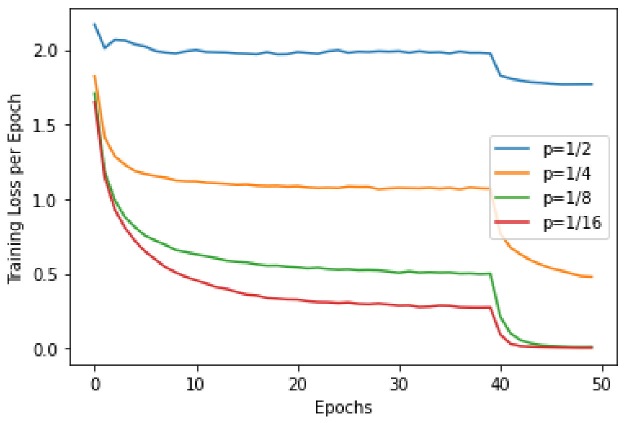

Figure 16. Training loss vs Epoch for Padam optimizer on CIFAR-10 using ResNet-18

From the graph of training loss vs Epochs for different values of p for the Padam optimizer, it is observed that the initial training loss is maximum for p = 0.5. As we increase the p value to 0.25, the observed initial training loss decreases. For p = 0.125 and p = 0.0625, we observe the lowest initial training loss. With increase in the number of epochs, training loss decrease for all the optimizers. After 40th epoch, with change in learning rate, the loss further decreases. p= 0.125 has the lowest final training loss for this ResNet-18 architecture.

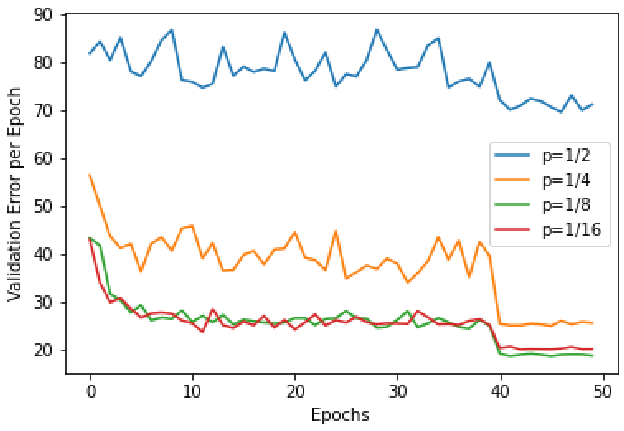

Figure 17. Validation loss vs Epoch for Padam optimizer on CIFAR-10 using ResNet-18

From the graph of Validation Error vs Epochs (ResNet-18) for different values of p for the Padam optimizer, it is observed that the initial Validation Error is maximum for p = 0.5. As we increase the p value to 0.25, the observed initial validation error decreases. For p = 0.125 and p = 0.0625, we observe the lowest initial validation error. With increase in the number of epochs, validation error decrease for all the optimizers. After 40th epoch, with change in learning rate, the validation error further decreases. Also, it is clearly evident that lower values of p lead to a higher convergence rate and also a better generalization. p= 0.125 has the lowest final validation error for this ResNet-18 architecture. Note: After observing the performance on the unseen test data, we observed test errors similar to the validation errors. Hence, the graphs are plotted for validation error instead of Test error as the validation dataset is the first point of an unseen data to the trained neural network model.

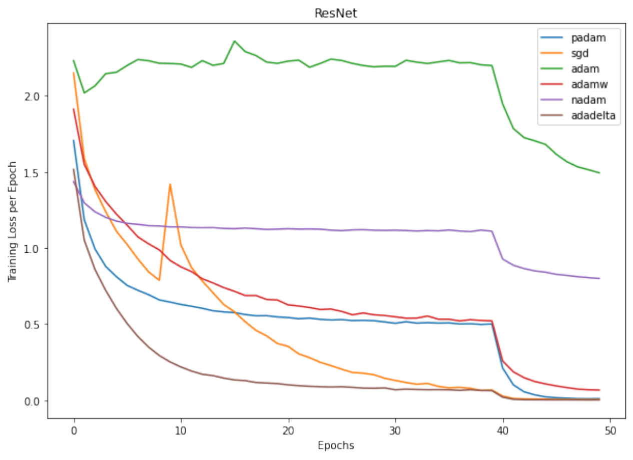

Figure 18. Train loss vs Epochs for different optimizers on CIFAR-10 using ResNet-18

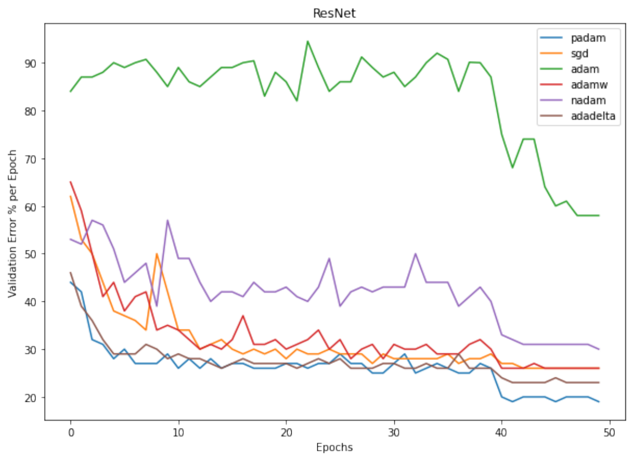

Figure 19. Validation loss vs Epochs for different optimizers on CIFAR-10 using ResNet-18

5.2. Comparison of performance of optimizers on CIFAR-10 for ResNet-18, VGGNet-16 and Wide ResNet

For ResNet-18, $p = 0.125$ resulted in the lowest validation error and fastest convergence to the optimum. Hence, optimum value of p is fixed at 0.125 for its comparisons with other sets of optimizers on 3 different Neural Network architectures: ResNet-18, VGG-16 and Wide ResNet. Training Loss is plotted with respect to epochs for different optimizers (with the optimum value of $p = 0.125$ for Padam). We observe that the Padam model has a faster convergence and a training loss as low as the SGD with momentum optimizer. Strangely, Adam optimizer for this set of hyperparameters performs poorly as compared to the other optimization techniques both in terms of convergence as well as generalization to the validation data.

Validation error is plotted with respect to epochs for different optimizers (with the optimum value of $p = 0.125$ for Padam). We observe that the Padam model starts with a low validation error and faster convergence besides stabilising to lower validation error after decrease in learning rate on the 40^{th} epoch.

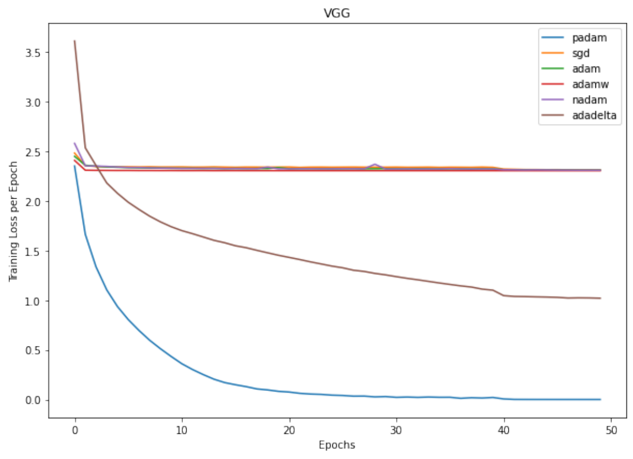

Figure 20. Train loss vs Epochs for different optimizers on CIFAR-10 using VGGNet-16

Next, we plot the training loss vs epochs for different optimizers and taking VGG-16 Neural Network architecture. Here, we can see that Padam optimization has significantly lower starting loss. Also, the model has a smooth descent towards the optimum and just after 20 epochs, training loss is stabilized to a very low value. Hence, for VGG-16 neural network architecture, for the selected hyperparameters, Padam optimizer has the best performance.

From the plot between the Validation Error vs epochs for

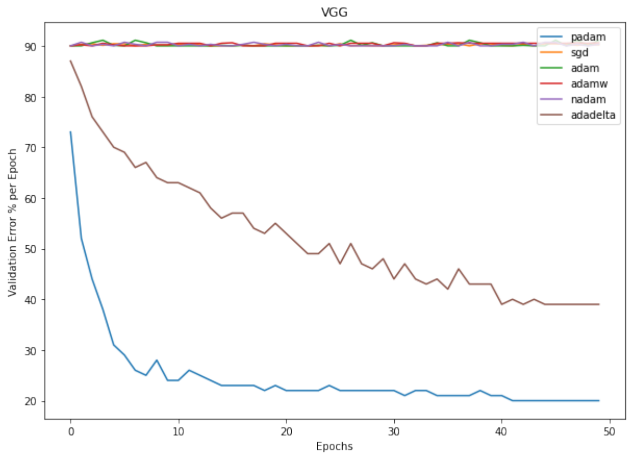

Figure 21. Validation loss vs Epochs for different optimizers on CIFAR-10 using VGGNet-16

the VGG-16 architecture, we clearly observe performance similar to that observed for the training loss. Also, the model has a smooth descent towards the optimum and just after 10 epochs, validation error is stabilized to a very low value. Hence, for VGG-16 neural network architecture, for the selected hyperparameters, Padam optimizer performs extremely well.

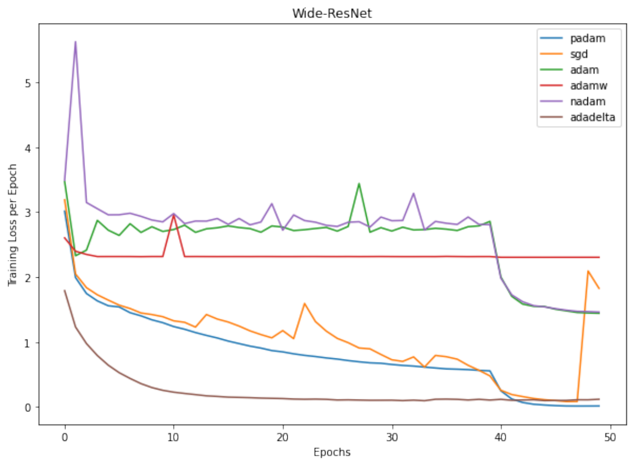

Figure 22. Train loss vs Epochs for different optimizers on CIFAR-10 using Wide ResNet

From, the above graphs for VGG-16, we can clearly see the superior performance of Padam optimizer (for a tuned value of p = 0.125) as compared to the other optimizers in terms of transit response for its descent to the optimum as well steady-state generalisation accuracy. Next, from the Validation loss graph for Wide ResNet architecture, we observe that Adadelta has a very fast convergence rate as it converges to it final validation error just after 15 epochs approximately. Padam although has a slower convergence rate as compared to the Adadelta, but still it has the best generalization error among all other optimizers. We observe a significant decrease in the validation loss after the 40^{th} epoch.

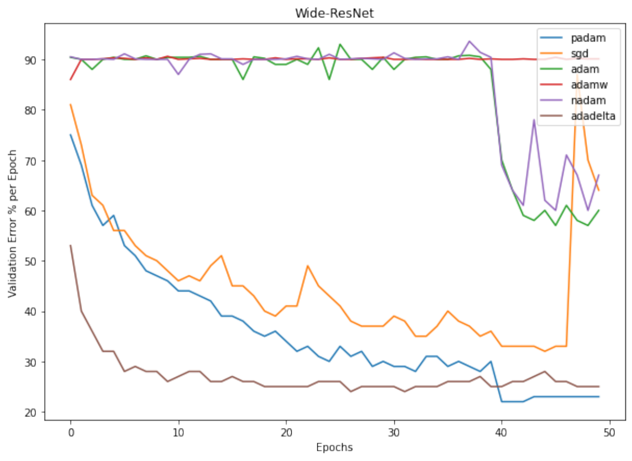

Figure 23. Validation loss vs Epochs for different optimizers on CIFAR-10 using Wide ResNet

From the Validation loss vs Epochs graph for Wide ResNet, we again observe that the Padam optimizer has a very fast convergence and a smooth descent towards the extremum. Adadelta, performs extremely well in terms of generalisations as well as convergence. However, Padam has the best generalization. SGD has fast but zig-zag convergence and also the validation error in case of SGD (with momentum) increases after approximately 50 epochs.

After observing the performance of different optimizers on CIFAR-10 image classification dataset, we can clearly notice significant improvement with the implemented Padam optimizer in the stochastic gradient approach towards the optimum.

5.3. Padam Optimizer-ResNet-18 Architecture on CIFAR-100 for multiple values of p

From the graph of training loss vs Epochs for different values of p for the Padam optimizer on CIFAR-100 dataset, it is observed that the initial training loss is maximum for p = 0.5. As we increase the p value to 0.25, the observed initial training loss decreases. For p = 0.125 and p = 0.0625, we observe the lowest initial training loss. With increase in the number of epochs, training loss decrease for all the optimizers. For p = 0.126 and p = 0.0625, the approach in reaching towards the optimum is identical. However, p

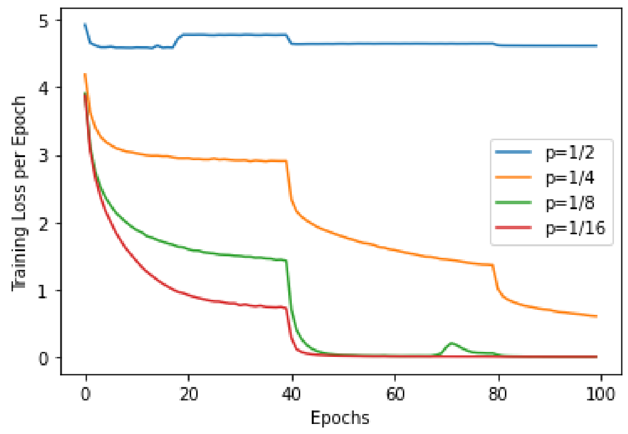

Figure 24. Training loss vs Epoch for Padam optimizer on CIFAR-100 using ResNet-18

= 0.0625 has a little smoother descent. After 40th epoch, with change in learning rate, the loss further decreases. p= 0.125 has the lowest final training loss for this ResNet-18 architecture.

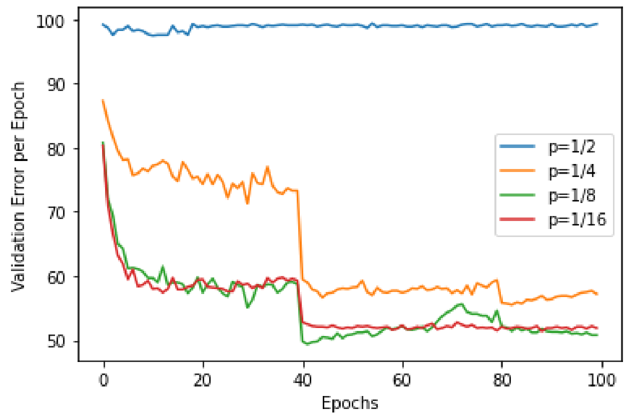

Figure 25. Validation loss vs Epoch for Padam optimizer on CIFAR-100 using ResNet-18

From the graph of Validation Error vs Epochs (ResNet-18) for different values of p for the Padam optimizer on CIFAR-100 dataset, it is observed that the initial Validation Error is maximum for p = 0.5. As we increase the p value to 0.25, the observed initial validation error decreases. For p = 0.125 and p = 0.0625, we observe the lowest initial validation error. With increase in the number of epochs, validation error decrease for all the optimizers. After 40^{th} epoch, with change in learning rate, the validation error further decreases. Also, it is clearly evident that lower values of p lead to a higher convergence rate and also a better generalization. p= 0.125 has the lowest final validation error for this ResNet-18 architecture. Note: Just like in CIFAR-10, in CIFAR-100 also, after observing the performance on the unseen test data, we observed test errors similar to the validation errors. Hence, the graphs are plotted for validation error instead of Test error as the validation dataset is the first point of an unseen data to the trained neural network model.

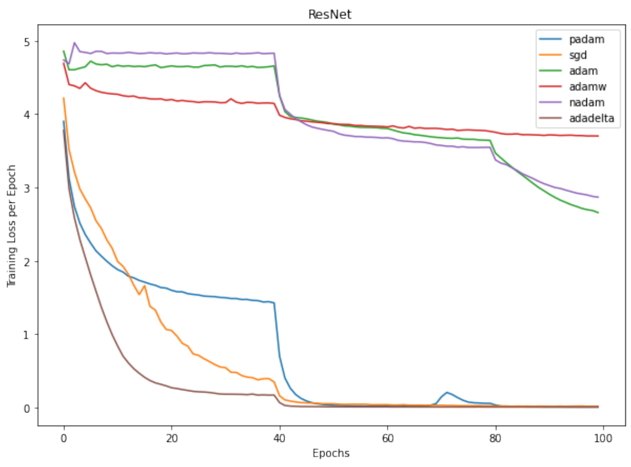

Figure 26. Training loss vs Epochs for different optimizers on CIFAR-100 using ResNet-18

5.4. Comparison of performance of optimizers on CIFAR-100 for ResNet-18, VGGNet-16 and Wide ResNet

For ResNet-18 and on CIFAR-100 dataset, just like CIFAR-10, it is observed that $p = 0.125$ resulted in the lowest validation error and fastest convergence to the optimum. Hence, optimum value of p is fixed at 0.125 for its comparisons with other sets of optimizers on 3 different Neural Network architectures: ResNet-18, VGG-16 and Wide ResNet. Training Loss is plotted with respect to epochs for different optimizers (with the optimum value of $p = 0.125$ for Padam) for ResNet-18 architecture. We observe that the Padam model has a faster convergence and a training loss as low as the SGD with momentum optimizer. Adadelta has the fastest decrease in the training loss as compared to the other optimizers .

Validation error is plotted with respect to epochs for different optimizers (with the optimum value of $p = 0.125$ for Padam) on the CIFAR-100 dataset. We observe that the Padam model starts with a low validation error and faster convergence besides stabilising to lower validation error after decrease in learning rate on the 40^{th} epoch.

For VGG-16 and on CIFAR-100 dataset, from the graph between the training loss and epochs, it is observed that

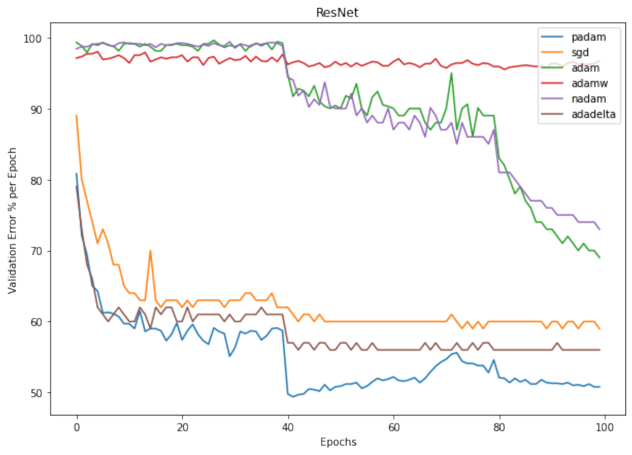

Figure 27. Validation loss vs Epochs for different optimizers on CIFAR-100 using ResNet-18

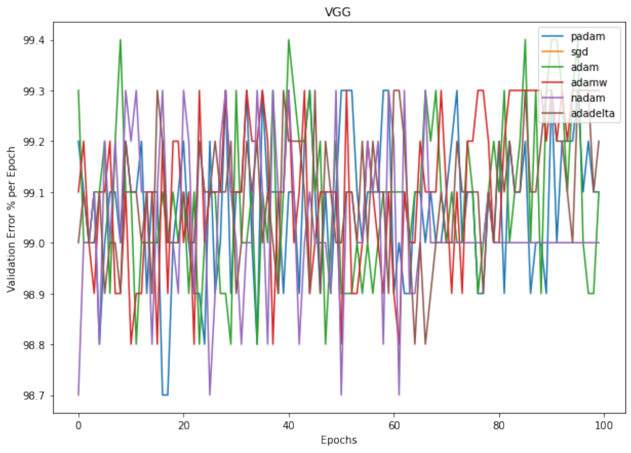

Figure 29. Validation loss vs Epochs for different optimizers on CIFAR-100 using VGGNet-16

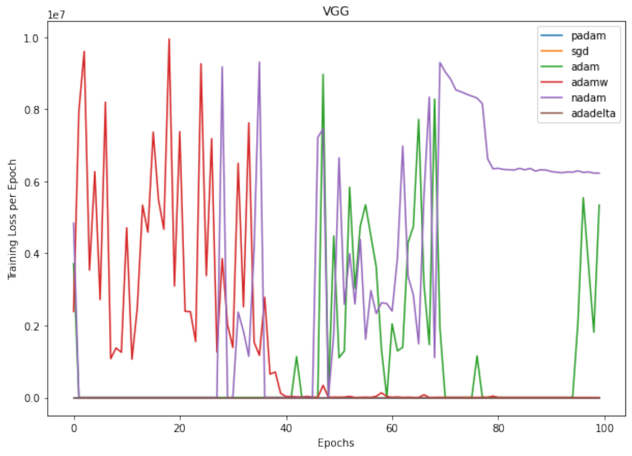

Figure 28. Training loss vs Epochs for different optimizers on CIFAR-100 using VGGNet-16

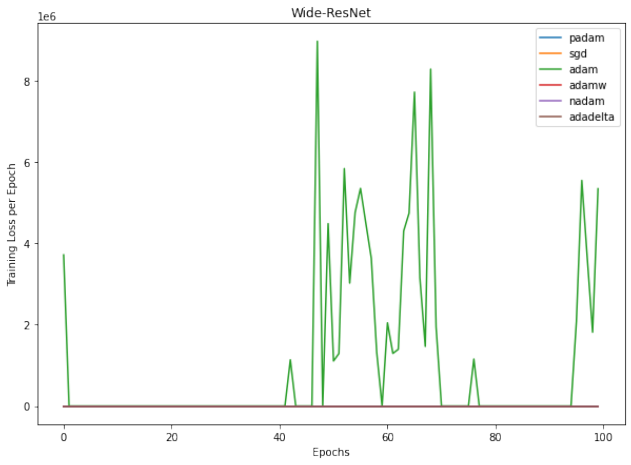

Figure 30. Training loss vs Epochs for different optimizers on CIFAR-100 using Wide ResNet

training loss is varying at a very high rate for almost all the optimization algorithms apart from the Adadelta optimizer. This problem may be related to the optimization and the hyperparameter settings for the neural network which might not be compatible with VGG-16 architecture on CIFAR-100 dataset since we are following baseline values for the optimal padam optimizer, we did not finetune these values.

From the Validation loss vs Epochs graph for Wide ResNet, we observe that the Padam optimizer has a very fast convergence and a relatively smooth descent towards the extremum. Adadelta, performs extremely well in terms of generalisations as well as convergence. However, Padam has the best generalization. SGD has fast but zig-zag convergence and also the validation error in case of SGD (with momentum) increases after 40 epochs.

6. Results and Inferences: Theoretical and Experimental

From the trends of validation Loss plots of Padam optimizer for different values of p on both the image classification datasets and by analysing the Test accuracy results from the test accuracy table of Padam with ResNet-18 architecture , we infer that, for p = 0.125, we achieve the highest test accuracy of 79% for CIFAR-10 and 49% for CIFAR-100. For CIFAR-10, it is also observed that as we decrease the p value from 0.25 to 0.0625, test accuracy increases from 74% to 79%. This trend is also observed for CIFAR-100 dataset, where test accuracy increases from

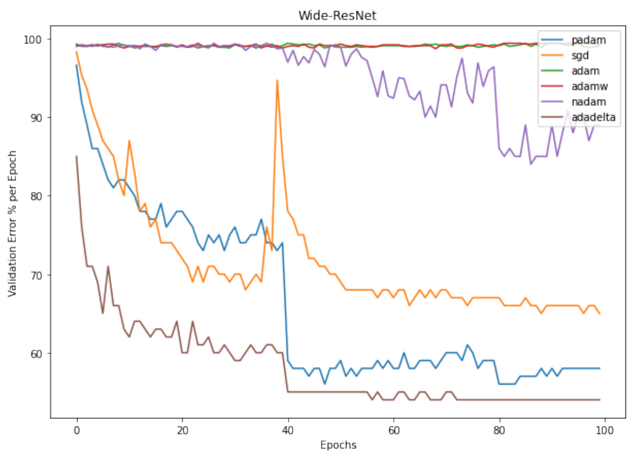

Figure 31. Validation loss vs Epochs for different optimizers on CIFAR-100 using Wide ResNet

42% at p = 0.25 to 48% at p = 0.0625. This reaffirms our reasoning that a better generalization is achieved as the p values are increased from p = 0 which is the case of SGD with momentum optimizer to p = 0.5 which is the same as AMSGrad/Nadam/Adam optimizer. Also, it is concluded that for this given set of model hyperparamters, there is only a small decrease in accuracy if the value of p reduces below 0.125. So, p = 0.125 is chosen as the optimal value because lowering p further will lead to slower rates of convergence. We also observed that this chosen optimal value of p, leads to improvement in convergence rate as well as generalisation for all the three neural network architectures i.e. ResNet-18, VGGNet-16 and Wide ResNet.

p	1/16	1/8	1/4
Test accuracy with ResNet-18 (%)	79.0	79.0	74.0

Figure 32. Test accuracy for Padam with different values of p on CIFAR-10 using ResNet-18

p	1/16	1/8	1/4
Test accuracy with ResNet-18 (%)	48.0	49.0	42.0

Figure 33. Test accuracy for Padam with different values of p on CIFAR-100 using ResNet-18

From the test accuracy table for different optimizers on CIFAR-10 and CIFAR-100 datasets, we can clearly infer that Padam optimizer achieves the best performance on generalization as compared to its counterparts. For ResNet-18 on CIFAR-10 dataset, Padam achieves 79% test accuracy followed by AdaDelta which achieved 76% test accuracy. We got 73% test accuracy for both Nadam and SGD with momentum optimizers. So, for Padam, in addition to getting the faster rate of descent to optimum, we were also able to achieve a better test accuracy as compared to our baseline SGD (with momentum) optimizer. For Wide ResNet architecture, the Padam optimizer achieved the maximum test accuracy of 78% on CIFAR-10 dataset. The point to note here is, that strangely for the base settings of model hyperparametrs, Nadam and SGD (with momentum) optimizers achieved very low test accuracy of 42% and 37% respectively.

For ResNet-18 on CIFAR-100 dataset, Padam achieves 49% test accuracy followed by AdaDelta which achieved 44% test accuracy. For Wide ResNet architecture, the Padam optimizer achieved the maximum test accuracy of 42% on CIFAR-100 dataset. Hence, from above observations we can clearly infer that changing the learning rate adaptive hyperparameter of p, not only leads to a faster convergence as compared to the SGD with momentum optimizer but also a better generalization than Nadam and AMSGrad optimizers. Also, as derived in the non-convex optimization complexity for Padam optimizers, a much smoother convergence with lower training time is attained. Note: Since, the main objective of this work was to compare the performance of Padam with respect to the other widely used optimizers for a base setting of model hyperparamters, we did not finetune the model to improve higher accuracy.

Models	SGD + Momentum	AdaDelta	Adam	AdamW	Nadam	Padam
ResNet-18	73.0	76.0	70.0	74.0	73.0	79.0
Wide ResNet	37.0	73.0	41.0	30.0	42.0	78.0

Figure 34. Test accuracy for different optimizers on CIFAR-10

Models	SGD + Momentum	AdaDelta	Adam	Nadam	Padam
ResNet-18	42.0	44.0	31.0	27.0	49.0
Wide ResNet	35.0	48.0	28.0	30.0	42.0

Figure 35. Test accuracy for different optimizers on CIFAR-100

6.1. Comparison with the original paper and New Results

Through our work, we were able to achieve a better generalization with the optimal value of p in Padam (p = 0.125) as compared to the SGD with momentum optimizer for all the three neural network architectures: ResNet-18, VGG-16 and Wide ResNet. Contrary to this, Jinghui et al. [1], achieved higher accuracy with SGD optimizer for the ResNet architecture. In addition to this, we also analysed, visualised and compared Padam optimization algorithm with the AdaDelta optimizer for CIFAR-10 image classification dataset. We achieved a good test accuracy of 76% and 73% with AdaDelta on ResNet-18 and Wide ResNet. Original paper did not analyse and compare the performance of Padam with AdaDelta. Also, in this work, we validated the derived complexity of non-complex objectives and the training times for the deep neural networks through our experimentation as well as by extensive mathematical analysis. In the earlier sections, we have, in detail, presented our observations, inferences, experimentation outcomes and analytical results.

7. Discussion and Conclusion

This work demonstrates that by changing the value of the hyperparameter p in the adaptive gradient optimizers, used in the adaptive learning term having the second order moment, we achieve the optimum in the two performance metrics: Convergence Rate and accuracy to the unseen data. Through the implementation of Padam algorithm, the problem of diminishing learning rate conundrum is solved with the help of an optimum p-norm (pth exponent) which gives a leeway to the model in terms of initial learning rate value selection. This trend can be interpreted from the validation loss plots of Padam, where a much smoother descent is observed with the loss changing only after the scheduled epochs. We optimize different combinations of hyperparameters for widely used optimization techniques and compare their performance with our implemented algorithm. Also, through our implementation, a guaranteed model convergence for non-convex objective functions is proposed and the model complexity for convergence is derived.

7.1. Future Scope

To take this work further, there is a scope in optimizing Padam optimizers for non-convex and non-concave objective functions on different model hyperparameter settings. Also, for the language models, where earlier work did not guarantee a significant improvement in performance with this optimization algorithm, there is a scope to tweak the adaptive term taking into account the sequential weights. Also, this technique can be used in Generative Adversarial Network (GAN) for achieving higher accuracy in the image batch generation.

References

[1] Chen, Jinghui Zhou, Dongruo Tang, Yiqi Yang, Ziyan Cao, Yuan Gu, Quanquan. (2020). Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks. 3239-3247. 10.24963/ijcai.2020/448. 3, 4, 16

[2] Zijun Zhang, Lin Ma, Zongpeng Li, and Chuan Wu. Normalized direction-preserving adam. arXiv preprint arXiv:1709.04546, 2017. 3

[3] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. arXiv preprint arXiv:1712.07628, 2017. 3

[4] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In International Conference on Learning Representations, 2018. 3

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. International Conference on Learning Representations, 2015. 3

[6] Mahesh Chandra Mukkamala and Matthias Hein. Variants of rmsprop and adagrad with logarithmic regret bounds. In International Conference on Machine Learning, pages 2545–2553, 2017. 3

[7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul):2121–2159, 2011. 3

[8] Amitabh Basu, Soham De, Anirbit Mukherjee, and Enayat Ullah. Convergence guarantees for rmsprop and adam in non-convex optimization and their comparison to nesterov acceleration on autoencoders. arXiv preprint arXiv:1807.06766, 2018. 3

[9] Xiaoyu Li and Francesco Orabona. On the convergence of stochastic gradient descent with adaptive stepsizes. arXiv preprint arXiv:1805.08114, 2018. 3

[10] Zhuang, Juntang Tang, Tommy Tatikonda, Sekhar Dvornek, Nicha Ding, Yifan Papademetris, Xenophon Duncan, James. (2020). AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients. 3

[11] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. In International Conference on Learning Representations, 2018. 3

[12] Yi Xu, Qihang Lin, and Tianbao Yang. Accelerate stochastic subgradient method by leveraging local growth condition. arXiv preprint arXiv:1607.01027, 2016. 3

[13] Rong Ge, Sham M Kakade, Rahul Kidambi, and Praneeth Netrapalli. The step decay schedule: A near optimal, geometrically decaying learning rate procedure. arXiv preprint arXiv:1904.12838, 2019. 3

[14] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In International Conference on Learning Representations, 2019. 3, 4

[15] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. In Proceedings of the 7th International Conference on Learning Representations, New Orleans, Louisiana, May 2019. 3

[16] Manzil Zaheer, Sashank Reddi, Devendra Singh Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In Advances in Neural Information Processing Systems, 2018. 3

[17] https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c 2

[18] https://medium.com/mlearning-ai/an-overview-of-vgg16-and-nin-models-96e4bf398484 7

[19] https://www.pluralsight.com/guides/introduction-to-resnet 7

[20] https://pytorch.org/hub/pytorch_vision_wide_resnet/ 7

[21] https://ruder.io/optimizing-gradient-descent 5, 6